Performing Speech Recognition on Multiple Parallel Files Using Continous Hidden Markov Models on an FPGA

Melnikoff, Stephen; Quigley, Steven; Russell, Martin

Performing Speech Recognition on Multiple Parallel Files Using Continuous Hidden Markov Models on an FPGA

S J Melnikoff, S F Quigley & M J Russell
Electronic, Electrical and Computer Engineering, University of Birmingham, Edgbaston, Birmingham, B15 2TT, United Kingdom
S.J.Melnikoff@iee.org,
S.F.Quigley@bham.ac.uk, M.J.Russell@bham.ac.uk

Abstract

Speech recognition is a computationally demanding task, particularly the stages which use Viterbi decoding for converting pre-processed speech data into words or sub-word unit, and the associated observation probability calculations, which employ multivariate Gaussian distributions; so any device that can reduce the load on, for example, a PC's processor, is advantageous. Hence we present two implementations of a speech recognition system incorporating an FPGA, employing continuous hidden Markov models (HMMs), and capable of processing three speech files simultaneously. The first uses monophones, and can perform recognition 250 times real time (in terms of average time per observation), as well as outperforming its software equivalent. The second uses biphones and triphones, reducing the speedup to 13 times real time.

1. Introduction

Real time continuous speech recognition is a demanding task which tends to benefit from increasing the available computing resources.

A typical speech recognition system starts with a pre-processing stage, which takes a speech waveform as its input, and extracts from it feature vectors or observations which represent the information required to perform recognition. The second stage is recognition, or decoding, which is performed using a set of phone-level statistical models called hidden Markov models (HMMs). In most systems, several context-sensitive phone-level HMMs are used, in order to accommodate context-induced variation in the acoustic realisation of the phone.

The pre- and post-processing stages can be performed efficiently in software (though some of the pre-processing may be better suited to a DSP). The decoding and associated observation probability calculations, however, place a particularly high load on the processor, and so it is these parts of the system that have been the subject of previous research using custom-built hardware. However, with ever more powerful programmable logic devices being available, such chips appear to offer an attractive alternative [6].

Accordingly, in this paper we describe our monophone and biphone/triphone implementations on an FPGA, of an HMM-based speech recognition system, which incorporates floating-point units for processing multivariate Gaussian distributions, along with a Viterbi decoder, to process three speech files simultaneously. This work follows on from [3] and [4], which dealt with single-file monophone implementations.

The paper is organised as follows. In section 2, we explain the motivation behind attempting to build a speech recogniser in hardware. In section 3, we explain the basic theory of speech recognition, with an overview of the implemented system in section 4. Sections 5 and 6 deal respectively with the theory of observation probabilities for continuous HMMs, and Viterbi decoding, detailing the design and implementation of the hardware in each case. The results are summarised in section 7, followed by conclusions.

2. Motivation

The ultimate aim of this work is to produce a hardware implementation of a speech recognition system, with an FPGA acting as a co-processor, that is capable of performing recognition at a much faster rate than software.

For most speech recognition applications, it is sufficient to produce results in real time, and software solutions that do this already exist. However, there are several scenarios that require much quicker recognition rates and so could benefit from hardware acceleration.

For example, in telephony-based call-centre applications (e.g. the AT&T "How may I help you?" system [1]), the speech recogniser is required to process a large number of spoken queries in parallel. There are also analogous non-real time applications, such as off-line transcription of dictation, where the ability of a single

0-7803-7574-2/02/$17.00 ©2002 IEEE

system to process multiple speech streams at high speed may offer a significant financial advantage.

Alternatively, the additional processing power offered by an FPGA might be used for real-time implementation of the "next generation" of speech recognition algorithms, which are currently being developed.

3. Speech recognition theory

The most widespread and successful approach to speech recognition is based on the hidden Markov model (HMM) [5][8], whereby a probabilistic process models spoken utterances as the outputs of finite state machines (FSMs). The notation here is based on [5].

The underlying problem is as follows. Given an observation sequence $O = O_0, O_1 \ldots O_{T-1}$, where each O_t is data representing speech which has been sampled at fixed intervals, and a number of potential models, each of which is a representation of a particular spoken utterance (e.g. word or sub-word unit), we would like to find the sequence of models which is most likely to have produced O. These models are based on HMMs.

An N-state Markov Model is completely defined by a set of N states forming a finite state machine, and an $N \times N$ stochastic matrix defining transitions between states, whose elements $a_{ij} = P(\text{state } j \text{ at time } t \mid \text{state } i \text{ at time } t-1)$; these are the *transition probabilities*.

In a hidden Markov model, each state additionally has associated with it a probability density function $b_j(O_t)$ which determines the probability that state j emits a particular observation O_t at time t (the model is "hidden" because any state could have emitted the current observation). The p.d.f. can be continuous or discrete; accordingly the pre-processed speech data can be a multi-dimensional vector or a single quantised value. $b_j(O_t)$ is known as the *observation probability*, and is described in more detail below.

Such a model can only generate an observation sequence $O = O_0, O_1 \ldots O_{T-1}$ via a state sequence of length T, as a state only emits one observation at each time t. Our aim is to find the state sequence which has the highest probability of producing the observation sequence O. This can be computed efficiently using Viterbi decoding (below).

Subject to having sufficient training data, the larger the number of possible utterances, and hence the larger the number of HMMs, the greater the recognition accuracy of the system.

4. System details

The complete system consists of a PC, and an FPGA on a development board inside it. For this implementation, the speech waveforms are processed in advance, in order to extract the observation data used for the decoding. This pre-processing is performed using the HTK speech recognition toolkit [7]. HTK is also used to verify the results produced by our system.

The speech data is sent to the FPGA, which performs the decoding, outputting the set of most likely predecessor states. This is sent back to the PC, which performs the backtracking process in software.

4.1. System hardware and software

The design is implemented on a Xilinx Virtex XCV2000E FPGA [11], sitting on Celoxica's RC1000-PP development board [9]. The RC1000 is a PCI card, whose features include the FPGA, and 8 Mb of RAM accessible by both it and the host PC. The RAM is configured as 4 banks of 2 Mb, each with a 32-bit data bus between it and the FPGA, and there is arbitration logic to prevent contention between the PC and FPGA.

The RC1000 sits inside a PC with a Pentium III 450 MHz processor. Our C++ software performs pre- and post-processing, and is also capable of carrying out all the same calculations as the FPGA, in order to compare performance, and to make it simpler to experiment with and debug the design during the development of the hardware version. The code is written so as to be as functionally similar to the FPGA implementation as possible.

In order to ensure uniformity of data between HTK and our software and hardware, our software uses the same data files as HTK, and produces VHDL code for parts of the design and for testbenches.

4.2. Speech data

The speech waveforms used for the testing and training of both implementations are taken from the TIMIT database [10], a collection of speech data designed for the development of speech recognition systems.

For these implementations, we use 49 monophone models for the first implementation, and 634 biphone and triphone models (i.e. pairs and triplets of monophones) for the second, all with 3 states, and no language model.

5. Observation probability computation

5.1. Theory

Continuous HMMs compute their observation probabilities $b_j(O_t)$ based on feature vectors extracted from the speech waveform. The computation typically uses uncorrelated multivariate Gaussian distributions [2].

Calculating values using the regular form of the equation would require significant resources if implemented in hardware with any degree of parallelism, as it requires multiplications, divisions and exponentations. Fortunately, as with Viterbi decoding, the process can be made more efficient if performed in the log domain:

$$\ln(N) = \left[-\frac{L}{2}\ln(2\pi) - \sum_{l=0}^{L-1} \ln(\sigma_{jl}) \right] - \sum_{l=0}^{L-1} (O_{tl} - \mu_{jl})^2 \cdot \left[\frac{1}{2\sigma_{jl}^2} \right]. \quad (1)$$

Note that the values in square brackets are dependent only on the current state, not the current observation, so can be computed in advance. For each vector element of each state, we now require a subtraction, a square and a multiplication.

5.2. Design

The block which computes the observation probabilities for continuous HMMs processes each observation's 39 elements one at a time, using a fully pipelined architecture. Due to the large dynamic range encountered during these calculations, the data values are processed as floating-point numbers.

A floating point subtractor, squarer and multiplier are used, with the resulting value sent to an accumulator. The output probability is then converted to fixed point and buffered, before being sent to the Viterbi decoder core.

Note that because the same observation data is used in the calculations for each state, these values need only be read in once for each time frame, freeing up part of the data bus for other uses. A buffer stores the values when they are read, then cycles through them for each HMM.

5.3. Implementation

The above design is implemented on the FPGA alongside the Viterbi decoder, with the observation, mean and variance data being read from off-chip RAM, one element of each per clock cycle. The constant in the first set of square brackets in equation (1) is treated as a fortieth element.

Because each observation probability depends on the sum of forty elements, a value is only written to the buffer once every forty cycles. The contents of this are sent to the decoder only when all of the HMMs' probabilities have been computed. As a result, the decoder sits idle for much of the time.

A convenient way of taking advantage of this spare processing time and the bandwidth freed up by only reading in the observation data once, rather than for each state, is to implement more observation probability computation blocks, operating in parallel on different observation data and the same model data.

For the monophone-based system, each observation probability computation block uses around 4400 LUTs (look-up tables) and 3700 FFs (D-type flip-flops). For the biphone/triphone system, each one uses 4500 LUTs and 5000 FFs. In both cases, the XCV2000E has sufficient resources to allow three such blocks to be instantiated.

While these could be used to process one speech file three times as fast, it was felt that in a real-world application, the speech data is more likely to be presented in real time, so three different files are processed at once instead. The files are read in and stored one after the other, and the model data delayed accordingly for the second and third blocks. They then take it in turns to use the decoder.

6. Viterbi decoding

Once the observation probabilities have been computed, we can proceed with the recognition process, as follows.

6.1. Theory

The arithmetic associated with Viterbi decoding mainly consists of multiplications and comparisons. By performing these calculations in the log domain, we can convert the multiplications into additions, which are more speed- and resource-efficient for implementation in hardware.

We define the value $\delta_t(j)$ as the maximum probability, over all partial state sequences ending in state j at time t, that the HMM emits the sequence $O = O_0, O_1 \ldots O_t$. It can be shown that this value can be computed iteratively – in the negative log domain – as:

$$\delta_t(j) = \min_{0 \le i \le N-1} [\delta_{t-1}(i) + a_{ij}] + b_j(O_t), \quad (2)$$

where i is a possible previous state (i.e. at time t–1).

This value determines the most likely predecessor state $\psi_t(j)$, for the current state j at time t, given by:

$$\psi_t(j) = \arg\min_{0 \le i \le N-1} [\delta_{t-1}(i) + a_{ij}]. \quad (3)$$

At the end of the observation sequence, we backtrack through the most likely predecessor states in order to find the most likely state sequence. Each utterance has an HMM representing it, and so this sequence not only describes the most likely route through a particular HMM, but by concatenation provides the most likely sequence of HMMs, and hence the most likely sequence of words or sub-word units uttered.

6.2. Design

The Viterbi decoder consists of five parts. The observation probabilities $b_j(O_t)$ enter through the initialisation and switching block, which sets the $\delta_t(j)$ values at the start of an observation sequence, and thereafter routes $b_j(O_t)$ and $\delta_t(j)$ to their respective destinations.

$\delta_t(j)$ is sent to two places. The scaler scales the probabilities, removing those corresponding to the least likely paths, hence preventing (negative) overflow. The language model block uses statistical information about the probability of one phone following another to compute each phone's most likely predecessor. In this particular

implementation, we are not using an explicit language model, so this block computes the single most likely predecessor for all phones, for each observation.

These values are then sent to the HMM processor, which contain nodes for implementing equations (2) and (3). As every node depends only on data produced by nodes in the previous time frame (i.e. at time $t-1$), and not the current one, we can – in theory – implement as many nodes as we like in parallel. In practice, however, three nodes (corresponding to the three states of one HMM) are implemented; while there is space for more to be processed in parallel, bandwidth limitations make this infeasible.

The nodes output the most likely predecessors of each HMM, $\psi_t(j)$, these values being written to RAM and processed in software, and the new $\delta_t(j)$ values.

These probabilities are sent to a buffer which provides space for the $\delta_t(j)$ values for all three speech files to be stored within the pipeline, before being sent back to the scaler.

6.3. Implementation

This part of the system is somewhat smaller than the observation probability computation blocks, using 1600 LUTs and 2800 FFs for the monophone version, while the biphone/triphone version uses 2500 LUTs and 2200 FFs.

Whereas the data for the observation probabilities is stored off-chip, the transition probabilities are stored in Block RAM, and the between-HMM probabilities (which form part of the language model block) are stored in distributed RAM.

7. Results

7.1 Monophone model

The full design occupies 74% of the XCV2000E's slices, requiring 16000 LUTs and 15000 FFs, and runs at 50 MHz.

The average time taken to process one observation is 39.3 µs. This compares to 5390 µs per observation for the software, making the hardware 137 times faster, and 10 ms for real time speech, a speedup of 254. As expected, this is three times faster than our previous implementation which only processed one speech file at a time.

7.2 Biphone and triphone model

With only the minimum of changes made to the original implementation, the initial version of this larger model required 143% of the FPGA's resources. After moving some of the buffers into Block RAM, this was reduced to 77%, including 22000 LUTs, 18000 FFs, and 138 Block RAMs (out of 160). The system runs at 33 MHz.

The average time per observation for the hardware is 769 µs, which is 95.6 times faster than software's 73.5 ms, and 13.0 times real time.

8. Conclusions

We have implemented a continuous HMM speech recognition system which uses an FPGA to compute the observation probabilities and perform Viterbi decoding for three speech files in parallel, using models based on 49 monophones, and 634 biphones and triphones.

The observation probability processing blocks compute values based on multivariate Gaussian distributions. They operate on floating-point data, and contain a total of six 24-bit multipliers and eighteen adders.

The Viterbi decoder processes three states simultaneously, and interleaves the three speech files under scrutiny.

The monophone system is capable of performing recognition over 130 times faster than a software equivalent, and 250 times faster than real time. For the biphone/triphone system running at a lower frequency, those figures are 13 and 96 times respectively.

References

[1] Gorin, A.L., Riccardi, G. & Wright, J.H., "How may I help you?" *Speech Communication*, **23**, No.1-2, 1997, pp.113-127.

[2] Holmes, J. N. & Holmes W.J., "Speech synthesis and recognition," Taylor & Francis, 2001.

[3] Melnikoff, S.J., Quigley, S.F. & Russell, M.J., "Implementing a hidden Markov model speech recognition system in programmable logic," *FPL 2001, Lecture Notes in Computer Science #2147*, 2001, pp.81-90.

[4] Melnikoff, S.J., Quigley, S.F. & Russell, M.J., "Speech recognition on an FPGA using discrete and continuous hidden Markov models," *FPL 2002, Lecture Notes in Computer Science #2438*, 2002, pp.202-211.

[5] Rabiner, L.R., "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proc. IEEE*, 77, No.2, 1989, pp.257-286.

[6] Stogiannos, P., Dollas, A. & Digalakis, V., "A configurable logic based architecture for real-time continuous speech recognition using hidden Markov models," *J. VLSI Signal Processing Systems*, 2000, **24**, No.2-3, pp.223-240.

[7] Woodland, P.C., Odell, J.J., Valtchev, V. & Young, S.J. "Large vocabulary continuous speech recognition using HTK," *ICASSP '94*, 1994, pp.125-128.

[8] Young, S., "A review of large-vocabulary continuous-speech recognition," *IEEE Signal Processing Magazine*, **13**, No.5, 1996, pp.45-57.

[9] http://www.celoxica.com/

[10] http://www.ldc.upenn.edu/Catalog/LDC93S1.html

[11] http://www.xilinx.com/